Future Generation Computer Systems 28 (2012) 1350–1362

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A coordinator for scaling elastic applications across multiple clouds

Rodrigo N. Calheiros *, Adel Nadjaran Toosi, Christian Vecchiola, Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 23 August 2011
Received in revised form
24 November 2011
Accepted 10 March 2012
Available online 21 March 2012

Keywords:
Cloud computing
Infrastructure as a service
Elastic applications
InterCloud

ABSTRACT

Cloud computing allows customers to dynamically scale their applications, software platforms, and hardware infrastructures according to negotiated Service Level Agreements (SLAs). However, resources available in a single Cloud data center are limited, thus if a large demand for an elastic application is observed in a given time, a Cloud provider will not be able to deliver uniform Quality of Service (QoS) to handle such a demand and SLAs may be violated. One approach that can be taken to avoid such a scenario is enabling further growing of the application by scaling it across multiple, independent Cloud data centers, following market-based trading and negotiation of resources. This approach, as envisioned in the *InterCloud* project, is realized by agents called *Cloud Coordinators* and allows for an increase in performance, reliability, and scalability of elastic applications. In this paper, we propose both an architecture for such Cloud Coordinator and an extensible design that allows its adoption in different public and private Clouds. An evaluation of the Cloud Coordinator prototype running in a small-scale scenario shows the effectiveness of the proposed approach and its impact on elastic applications.

1. Introduction

Cloud computing [1] revolutionized the way computational resources are commercialized and delivered to customers. Built on top of technologies such as virtualization and service-oriented architectures, Cloud computing allows customers to dynamically scale their applications, software platforms, and hardware infrastructure, accordingly to agreed Service Level Agreements (SLAs).

Cloud resources can be acquired in different abstraction levels. Of special interest in this paper is the level known as *Infrastructure as a Service* (IaaS), where customers lease hardware resources (typically, but not restricted to, virtualized resources) in the provider's data center. These resources are directly accessed by the customers, which install and configure the whole software stack in order to meet their specific needs. From the customers point of view, a "sufficiently" large number of resources are available, and they are acquired and released according to the actual demand. These capabilities make Cloud computing a viable alternative to traditional provisioning methods where capacity planning is based on the peak demand, which is observed only in specific times and thus leads to low efficiency in resources utilization.

Nevertheless, resources available in a single data center are limited: even though data centers may contain thousands of physical machines able to host tens of thousands of virtual machines each, a large demand may put pressure in the data center capacity, especially in the case of private Clouds; thus, data center resources will be oversubscribed, resulting in differences in the performance obtained from resources that should deliver similar performance [2,3], or SLA violations. Moreover, in such a situation elastic applications hosted in such a stressed data center may not be able to scale up, and this may also result in SLA violations.

One approach that can be taken to counter the above situation is enabling applications to scale across multiple, independent Cloud data centers following market-based trading and negotiation of resources between providers and brokers. This approach, proposed in the *InterCloud* project [4], comprises Cloud data centers and brokers that dynamically negotiate resources between themselves in order to seamlessly meet elastic applications' SLAs by scaling applications across various data centers. InterCloud envisions market-oriented policies for provisioning of virtual machines across multiple data centers that can be adopted solely or together with ad-hoc policies adopted by specific providers. Because data centers geographically dispersed can be deployed for this purpose, it helps increasing application *reliability*; dispersed providers also allow application users to be served by resources that are closer to them, reducing communication latency and increasing application *performance*; finally, because more sources of resources are available, applications are able to support more users, improving application *scalability*.

Key components of InterCloud are the *Cloud Coordinator*, which represents providers in the marketplace, and the *Cloud Exchange*, which acts as a discovery system and offers a publication system.

* Corresponding author. Tel.: +61 3 83441347.
E-mail address: rncalheiros@ieee.org (R.N. Calheiros).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.03.010

In this paper, we focus on the architecture and design of the Cloud Coordinator. Our previous works focused on aspects such as market making [5], negotiation protocols [6], and decision on the moment to buy and sell resources and the price for them [7]. This work is built on top of these findings, and demonstrates how these aspects can be leveraged in the InterCloud scenario.

The Cloud Coordinator allows providers[1] to trade resources in response to changes in elastic application's demands or to user requests. In this sense, it acts as a trading agent [8]. However, the Cloud Coordinator also carries out other tasks: its backend contains a virtual machine management integration layer that allows utilization of available virtual machine management technologies, and its frontend is composed of a web service that communicates with other Cloud Coordinators and a Market Engine that evaluates offers and resources. Because Cloud Coordinator has information about the local infrastructure and the corresponding utilization rate, as well as access to other Coordinators, this is the component where policies for trading of resources must be implemented.

Contributions of this paper are threefold. First, it proposes an architecture of the Cloud Coordinator. Second, it proposes an extensible design that allows adoption of the Cloud Coordinator in different public and private Clouds. Third, it presents a prototype implementation of Cloud Coordinator running in a small-scale scenario that demonstrates the feasibility of the proposed architecture and its impact on elastic applications.

The rest of this paper is organized as follows. Section 2 discusses related work in the context of Grid computing, federation, and markets for computing resources. Section 3 presents a background overview of the InterCloud project and its key components. Section 4 presents the proposed architecture and design of the Cloud Coordinator component of InterCloud. Section 5 contains an evaluation of the Cloud Coordinator prototype running in a small-scale InterCloud scenario. Finally, Section 6 concludes the paper and presents future research directions.

2. Related work

Works that share the vision and goals of InterCloud are found in the area of Grid computing, Grid and Cloud federation, and markets for computational resources.

2.1. Grid computing

In Grid computing [9] systems, user-level applications are managed by resource brokers, which are in charge of selecting and allocating suitable resources for the application, and for scheduling tasks that compose the application in the selected resources. Allocation on such systems is typically handled in time slots, and the broker defines the allocation time of resources based on the application expected execution time. Because Grids are used by virtual organizations with predefined agreements about conditions for resource utilization, discovery of new resource providers may be accomplished off-line: once a new member joins the virtual organization, brokers can be configured to query new members' resource managers.

Clouds, however, differ from Grid system in several aspects, such as: (i) utilization of Cloud resources incur financial compensation to the Cloud provider, and different providers impose different policies for pricing their resources; (ii) resource allocations in Clouds are not time-based, and explicit deallocation of resources are necessary; and (iii) providers typically operate independently. The latter aspect complicates discovery of new potential providers of Cloud resources for the elastic application managed by the broker. Therefore, in this paper we propose a different approach for enabling elastic applications where resource discovery and acquisition for elastic applications is performed by the data center hosting the application, rather than by the broker.

2.2. Grid and cloud federation

InterGrid [10] applied virtualization technology to allow resource sharing among Grids. Resources where sought in remote Grids whenever an incoming allocation request could not be served by local available resources. Grid and Cloud resources are typically differently allocated. While a Grid federation is typically based in cooperation and sharing of resources with time-constrained allocations, Cloud allocations are typically not timely constrained (customers explicitly request resources release) and are based on financial compensation from customers to resource providers. Therefore, solutions for federation aimed at Grid platforms cannot be directly applied to Clouds, and thus cannot be directly applied in the context of InterCloud.

Several platforms for Cloud federation were proposed recently, with different motivations and incentives for parties joining the federation. Reservoir [11] introduces a modular and extensible Cloud architecture that supports business service management and Cloud federation. Claudia [12] provides an abstract layer enabling execution of services on top of a transparent federation of Cloud providers. However, these approaches do not consider either the mechanisms for formation of federation or specific discovery system, thus the approach presented in this paper can be leveraged by these systems to enhance their functionalities.

Open Cirrus [13] is a closed federation between universities and research centers in order to aid research in design, provisioning, and management of services in scale of multi data centers. Because Open Cirrus offers a closed federation, it is not possible for data centers to join and leave the group, and market mechanisms are not necessary.

Sky Computing [14] introduces a virtual site overlayed on dynamically provisioned distributed resources of several data centers and a closed federation model, where sharing of resources are based on cooperation (like in Grids) and not on market incentives as in InterCloud.

OPTIMIS [15] is a platform for Cloud service provisioning that manages the whole lifecycle of the service – from construction to execution – and also addresses issues such as risk and trust management, energy efficiency, and legislation. However, OPTIMIS does not addresses negotiation and a marketplace for discovery and negotiation of resources, as does InterCloud. Therefore, OPTIMIS and InterCloud complement each other, as each one provides features not available in the other.

Other works focus on low-level aspects of federation, such as security, networking, and image management [16,17]. These works complement the efforts presented in this paper, which encompasses higher-level aspects that are also necessary in federations like discovery and publication of offers and requests and policies for resource sharing.

Moreover, none of the previously discussed works share the goals of the InterCloud project—namely improve performance, reliability, and scalability of elastic applications by leveraging resources from multiple Clouds.

2.3. Markets for computing resources

Works related to systems for market-making such as GridEcon [18], SORMA [19], and Mandi [5], as well as the works by Song et al. [20], Mihailescu and Teo [21], Gomes et al. [22], and Vanmechelen et al. [23] concern mechanisms for creating markets,

[1] Throughout this paper, we use the term provider to refer to any type of data centers that offer IaaS services.

trading resources (e.g., auctions or fixed price), and also mechanisms to motivate market maintenance, i.e., study and development of techniques that motivate both resource providers and resource consumers to join and stay in the market.

Other works focus on the customers interaction with Grid and Cloud markets. In these works, policies are defined so that execution of applications is optimized either in terms of execution time or in terms of budget expenditure. Moreover, applications may contain dependences related to the order of execution of tasks. Typically, decision about execution ordering, allocation of resources among one or multiple resource providers, and scheduling of tasks on these resources are made by brokers. This concept was further extended so that brokers do not represent single users; instead, brokers represent a group of users of some facility that needs to expand resource capacity by buying resources from a Grid or Cloud. This is the case of works by Kim et al. [24], Ostermann et al. [25], Assunção et al. [26], Tordsson et al. [27], and Bossche et al. [28]. These works focus on policies for the resource consumer, while this work focuses in the resource providers.

Most of the works discussed so far derived from Grid computing works. Therefore, motivation for users to join the market relates to accelerating execution of high-performance or high-throughput scientific applications. Recently, with the advent of Clouds, another scenario for resource acquisition in markets is being researched. Such a scenario comprises a SaaS provider that offers services to users, and resources to offer such services are leased from IaaS providers. This is the case of the works by Fitó et al. [29] and Lee et al. [30]. These works differ from the works related to brokers by having different parameters acting as constraints and goals (SLAs and profit instead of budget and execution time). Nevertheless, these works also focus on policies for selection of providers and acquisition of resources, and not for resource provisioning in the market.

Regarding market policies for resource providers, Goiri et al. [31] presents a profit-driven policy for decisions related to insourcing and outsourcing resources. However, such a policy presents only general guidelines to decide when to perform the actions. It neither considers how to select resource suppliers, nor solves the problem of discovering such suppliers, as does our work.

3. InterCloud: an overview

Fig. 1 depicts the general organization of InterCloud, which envisions a marketplace that enables brokers and providers to improve performance, reliability, and scalability of elastic applications by leveraging resources from multiple Clouds in order to seamlessly meet applications' SLAs by scaling them across various data centers. InterCloud envisions market-oriented policies for provisioning of virtual machines across multiple data centers that can be adopted solely or together with *ad hoc* policies enforced by providers.

Central to the model is the presence of the *Cloud Exchange*. This component of InterCloud acts as a market-maker by supplying registry, negotiation, and founding services. Providers joining InterCloud query the Cloud Exchange to discover other parties to negotiate with. Negotiation for resources is handled directly by the involved parties, and it is not mediated by the Cloud Exchange. Alternatively, the Cloud Exchange can host auctions for resources by leveraging the Mandi architecture [5]. The presence of a market-maker component allows static pricing policies (as currently adopted by most of the Cloud providers) to be replaced by dynamic pricing policies, which can increase profit of providers [31,23,22].

Resource offers and requests are described in terms of number and characteristics of available virtual machines. Providers with spare resources, or demand for resources, send the following information to the Cloud Exchange (other parameters, such as I/O performance, latency, reputation, and location may be adopted over time, without significant changes in the general processes and methods presented here):

- Number of available/required virtual machines;
- Amount of memory of these machines;
- Number of cores of each machine;
- Computational capacity of each core, in Amazon EC2 Computing Units [32] or other agreed metric for measuring CPU capacity;
- Price of each virtual machine (per hour).

Discovery of available offers and requests is made via specific services offered by the Cloud Exchange.[2] Parameters of the service are characteristics of the required virtual machines, and the return of the query is the addresses of Cloud Coordinators that have requests/offers that match the corresponding offers/requests. A summary of services offered by the Cloud Exchange, their input parameters and corresponding actions is presented in Table 1.

Bid for resources and resource offers are published in the Cloud Exchange with *putResourceOffer* and *putResourceBid*. Publishers receive as result an id that is used for removing the offer/bid when the asset is not available (via *removeResourceOffer* and *removeResourceBid* services). If a publisher wants to remove all the offers or resource bids it published, services *removeAllOffers* and *removeAllBids* are used. Finally, the services *getCurrentOffers* and *getCurrentBids* enable searches for offers and bids that satisfy the given resources specification.

Providers access the Cloud Exchange and other parties via the *Cloud Coordinator*. The Cloud Coordinator frontend implements a service-oriented architecture to receive requests from other parties and also contains a service client that can make requests to other parties. The Cloud Coordinator backend implements a virtual machine manager abstraction layer, therefore the Cloud Coordinator is independent from the specific management technology used in the data center.

Resources traded in the market are virtual machines. Parameters such as number and capacity of cores and memory are used to define prices of instances. No specific market mechanism is imposed by InterCloud. Therefore, each provider defines the price for resource utilization and applies its own billing mechanisms.

Virtual machine images are assumed to have been created beforehand in the resource provider Cloud. Currently, it is required because specificities on each Cloud data center infrastructure may require some specific kernel modules and other low-level software, which are bundled together with customer software in the VM image. Even though there are efforts, such as the Open Virtualization Format (OVF) [33], towards standardization of virtual machines images, they are still not widely adopted. Therefore, measures are necessary that enforce software compatibility in different providers, such as our assumption of a pre-created image.

This paper focus on the architecture and design of the Cloud Coordinator as an agent for supporting elastic applications in the Cloud, so that user's brokers do not have to directly discover and acquire resources from multiple data centers for the application. However, the same mechanisms proposed for the Cloud Coordinator can be implemented in the broker to allow it to directly interact with multiple Cloud providers. Alternatively, similar mechanisms can be implemented by independent providers willing to have a more customized agent able to negotiate in the InterCloud market.

[2] Even though current services offered by this component are directory services, this component defines interfaces for auctioning services. Design and implementation of the Cloud Exchange and its market features are out of scope of this paper.

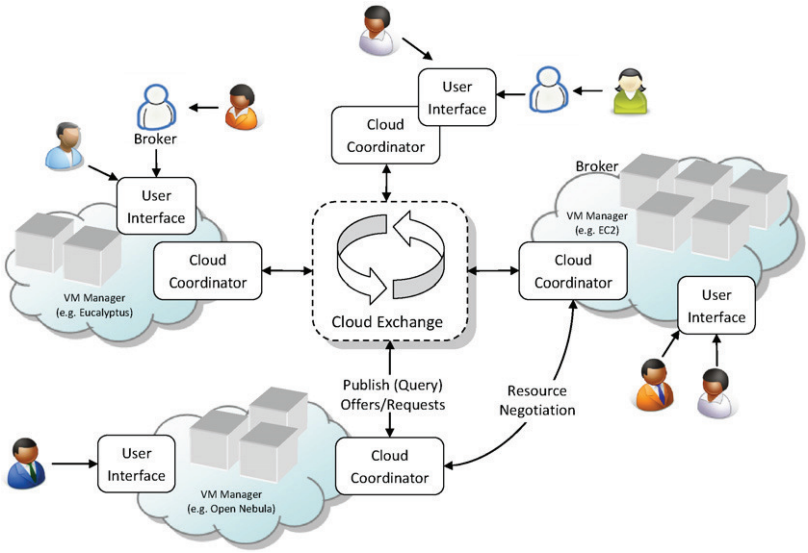


Fig. 1. InterCloud components and their roles in the architecture.

Table 1
Summary of services offered by the Cloud Exchange, their input parameters and resulting action.

Service name	Parameters	Action
putResourceOffer	VMs, memory, cores, computing capacity, price	Stores the offer and returns its corresponding id
putResourceBid	VMs, memory, cores, computing capacity, price	Stores the request and returns its corresponding id
getCurrentOffers	VMs, memory, cores, computing capacity, price	Returns addresses and offers of Cloud Coordinators that match the input request
getCurrentBids	VMs, memory, cores, computing capacity, price	Returns addresses and requests of Cloud Coordinators that match the input offer
removeResourceOffer	Id of the offer	Removes offer represented by id (if published by the caller Cloud Coordinator)
removeResourceBid	Id of the request	Removes request represented by id (if published by the caller Cloud Coordinator)
removeAllOffers	–	Removes all offers published by the caller Cloud Coordinator
removeAllBids	–	Removes all requests published by the caller Cloud Coordinator

4. Cloud Coordinator architecture and design

The Cloud Coordinator is the element that has to be present on each data center that wants to interact with InterCloud parties. The Cloud Coordinator is also used by users and brokers that want to acquire resources via InterCloud and do not own resources to negotiate in the market. In this case, the Cloud Coordinator can be seen as a data center that has no available local resources, and thus it always buys resources in the InterCloud marketplace in response to changes in the elastic application demand.

The Cloud Coordinator also offers services so that other parties can negotiate resources/requests and also access services offered by the Cloud Exchange and other Cloud Coordinators. It also interfaces with the rest of the data center components so resources are bought and sold according to the data center demand. In this sense, the Cloud Coordinator acts as a trading agent [8], even though its capabilities are not restricted to those of a trading agent. In fact, the Cloud Coordinator contains a trading agent, but it also carries out tasks related to virtual machine management and requests processing. The Cloud Coordinator architecture is presented in Fig. 2 and its class diagram is depicted in Fig. 3. In the rest of this section, we describe each of the Cloud Coordinator components.

4.1. Cloud Coordinator Server Proxy

The Cloud Coordinator Server Proxy is the component that implements communication with other coordinators. This is a web service whose methods implement the Alternate Offers protocol for resource negotiation [6] and also services for access to resources sold or bought. Every time a service request from another party is received, the Cloud Coordinator Server Proxy forwards it

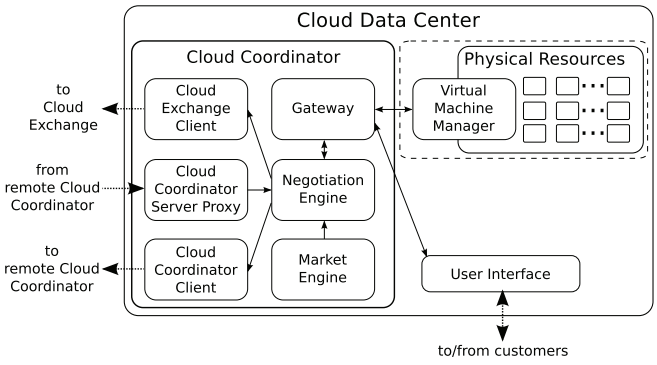


Fig. 2. Cloud Coordinator components. Elements surrounded by dashed lines are optional.

to the Negotiation Engine, which takes the proper action related to the request. Table 2 presents a summary of services offered by the Cloud Coordinator[3] and that are processed by the Proxy, their input parameters and corresponding actions.

4.2. Cloud Coordinator and Cloud Exchange Clients

The Cloud Coordinator Client and the Cloud Exchange Client implement clients for accessing the corresponding remote servers

[3] The exposed service name for parties interacting with the CloudCoordinator Server Proxy is "CloudCoordinator". For the sake of simplicity, we refer to the exposed service name rather than the name of the internal component, in the discussion throughout this paper.

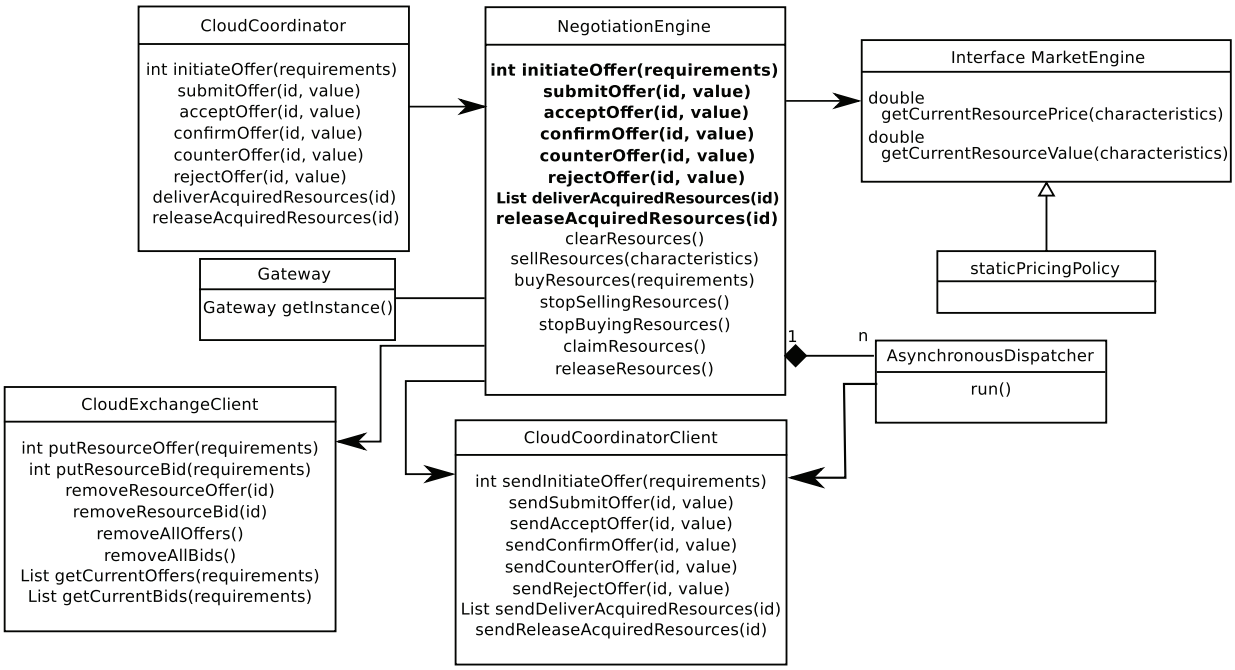


Fig. 3. Cloud Coordinator class diagram containing its main supporting classes and their public methods. Method names in bold are methods offered in the form of web services.

Table 2
Summary of services offered by the Cloud Coordinator, their input parameters and resulting action.

Service name	Parameters	Action
initiateOffer	operation (buy or sell) VMs, memory, cores, computing capacity	returns negotiation id
submitOffer	negotiation id, offer	Web services implementation of actions related to the Alternate Offers Protocol [6]
acceptOffer	negotiation id, offer	
counterOffer	negotiation id, offer	
rejectOffer	negotiation id, offer	
confirm offer	negotiation id, offer	
deliverAcquiredResources	negotiation id	Returns the list of addresses of VMs acquired in the given negotiation
releaseAcquiredResources	negotiation id	Releases acquired resources

in InterCloud. To be able to publish offers and requests, and also to be able to make queries to the Cloud Exchange, the Cloud Exchange Client has to be configured with the Cloud Exchange endpoint address. This information is supplied via a configuration file read at time of the Cloud Coordinator initialization.

Regarding Cloud Coordinator clients, they are created dynamically every time a negotiation process starts. If the negotiation process starts locally, the remote Cloud Coordinator address is obtained from the response of the query sent to the Cloud Exchange. If the negotiation process starts remotely, a Cloud Coordinator Client is created in response to a *initiateOffer* request received by the remote Cloud Coordinator. The new client endpoint is the endpoint of the sender of the request.

In either case, Clients exist only for the duration of interaction between coordinators: in case of failure in negotiation, client endpoints are terminated when the negotiation finishes. If negotiation succeeds, endpoints are kept for the time that one party is using resources from the other, and they are terminated when resources are released by the buyer. This avoids that references to parties that left InterCloud are used inadvertently in the system.

4.3. Negotiation Engine

The Negotiation Engine is the Cloud Coordinator component that mediates interaction between Cloud Coordinators. It not only processes requests received from remote Cloud Coordinators but also processes and negotiates remote resources required by local customers. Every time a customer request for resources arrives in the local Cloud (via Gateway), this component updates availability information. Policies for determining when providers should buy or sell resources are discussed in our previous publication [7]. Here, we focus on how to enable the process in practice with the help of Cloud Coordinator.

The Negotiation Engine also processes requests from other Coordinators (which are received via a Cloud Coordinator server proxy), and proceeds with negotiation via Alternate Offers protocol. In the beginning of the negotiation process, the Coordinator providing resources reserves them until the end of the negotiation, to avoid overallocation of resources among concurrent negotiations.

Fig. 4 depicts the process of publishing and negotiation of resources in InterCloud. Cloud Coordinators wanting to publish a resource offer (Cloud Coordinator A in the figure) or a resource request (Cloud Coordinator B in the figure) publish the offer/request via their Cloud Exchange Clients. Optionally, Cloud Coordinators query the Cloud Exchange about availability of requests/offer that match their request. If such a match exists, the list of matches and the Cloud Coordinator that published them is returned.

Because resource availability of providers are highly dynamic, it can change after publication of availability in the Cloud Exchange,

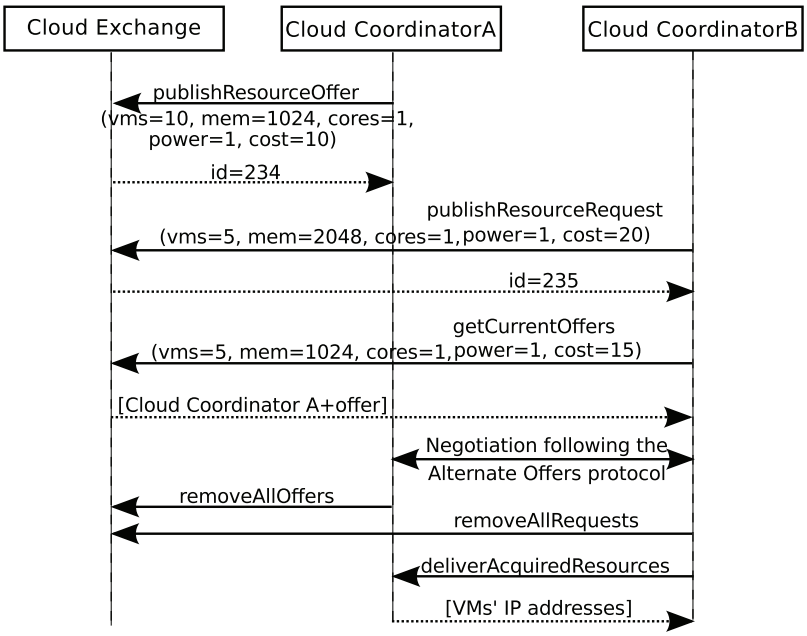


Fig. 4. Negotiation process in InterCloud. Cloud Coordinator A's data center has available resources and wants to sell them. Cloud Coordinator B's data center needs resources and wants to buy them. In this figure, we assume a successful negotiation between parties.

and thus query information may be out of date. Therefore, price information provided by the Cloud Exchange is used for deciding the order in which providers are contacted. However, negotiation runs between both Cloud Coordinators without necessarily relying on the price given by the Cloud Exchange. Communication with remote Cloud Coordinators is handled by Cloud Coordinator clients that are dynamically created to interact with each InterCloud member.

Negotiation between coordinators follows the Alternate Offers protocol [6]. In this protocol, the initiator party contacts the other party to give an offer to buy or sell resources. Receiver of the message can accept the offer, reject it, or present a counter offer. In the latter case, decision on which of the three actions is taken is passed to the other party. The process continues until parties agree with the offer or one of them rejects the offer and leaves the negotiation. The logic that drives the negotiation process is presented in Algorithm 1.

Based on the message received from the remote peer (i.e., accept, reject, or counter), the type of operation (buy or sell) and the offer received previously, Negotiate Engine decides whether offer is accepted, rejected, or countered. It is accepted when the other party is willing to pay for a local resource a value that is equal to or bigger than the current value set by the Market Engine. If the other party is selling resources, offer is accepted when the resource price is equal to or smaller than the value given by the local Market Engine.

If the condition for acceptance was not reached, but the other party updated its offer towards the condition being accepted (e.g., the price for selling the resource has been reduced by the seller since the last offer, but still does not match the maximum price paid by the buyer), a counter message is sent with the required value. Finally, if the party did not improve its offer, a "reject offer" message is sent and the negotiation finishes. Once the accept message is sent by some party, the rest of the protocol (e.g., confirmation from the other party) is executed. The Alternate Offers has been shown as being symmetric [6] in the sense that both parties in the negotiation process are equally able to make proposals, accept, and reject offers from other parties. This allows Coordinators to abort negotiation when the other party is not improving its offers, is delaying the negotiation process, is trying to manipulate the process, or if the Coordinator loses interest in a specific negotiation process (for example, if the negotiation price becomes unattractive). Similarly, if for any reason the resources being negotiated become unavailable, the negotiation process is aborted by the seller.

Because communication between Cloud Coordinators is based on web services that are synchronous, and because the negotiation process requires asynchronous messages to be sent between parties, the Negotiation Engine contains a pool of Asynchronous Dispatcher threads that handle submission of asynchronous messages to remote Cloud Coordinators. Such asynchronous messages are the messages defined in the Cloud Coordinator interface, except the two messages that have return values, namely *initiateOffer* (which returns negotiate id) and *deliverAcquiredResources* (which returns list of addresses of acquired resources). Sending of these two messages is performed synchronously through direct communication between the Negotiation Engine and the Cloud Coordinator Client.

Currently, for a Cloud provider to be able to use resources from another provider, it is required the availability, in the resource seller side, of a virtual machine image compatible with the resource seller's environment, and configured with the resource buyer's software. Because current focus of the project is in the market formation and negotiation, Virtual Machine creation and transferring is not handled by InterCloud.

If the negotiation succeeds, resources are made available to the buyer, who requests them via a specific service. When resources are no longer required, a release service is invoked by the buyer. This request is then forwarded to the VM manager, which takes the relevant actions to release the resources and, if it is the case, to republish their availability to the Cloud Exchange. Once resources are requested by the buyer, their virtual machine instances are deployed in the local data center and these VMs can be used by the buyer's consumers just like other resources are used.

The operation described so far details current negotiation steps performed by the Cloud Coordinator, which is based in the Alternate Offers protocol [6]. Other negotiation processes and strategies, such as auctions [5], futures and spot markets [23],

Algorithm 1: Decision-making process during resources negotiation between Cloud Coordinators. Operation type is defined by the party that sends the message, whereas decision is made by the party that receives the message. Therefore, BUY means that the sender of message wants to buy resources and SELL means that the sender wants to sell resources. Similarly, offer type is also defined by the sender based in Alternate Offers operations.

Data: offer: Alternate Offers message received from a remote Cloud Coordinator.
Data: requiredResource: description of resource under negotiation, defined in the *initiateOffer* message received in the beginning of negotiation.

```
1  resourceValue ← MarketEngine.getValue(resourceSpecification);
2  if offer.type = SUBMIT then
3      if (offer.operation = BUY ∧ offer.value≥ resourceValue)∨(offer.operation = SELL ∧ offer.value≤ resourceValue) then
           /* good offer: accept it.                          */
4          send ACCEPT;
5      end
6      else
           /* not very good offer: try to negotiate.          */
7          send COUNTER(resourceValue);
8      end
9  end
10 else if offer.type = COUNTER then
11     if (offer.operation = BUY ∧ (offer.value < resourceValue ∧ offer.value > lastOffer)) ∨ (offer.operation = SELL ∧ (offer.value > resourceValue ∧ offer.value < lastOffer)) then
           /* party upgraded its offer, but it is still not good enough: try to negotiate. */
12         send COUNTER(resourceValue);
13     end
14     else
           /* party did not upgrade its offer: reject.        */
15         send REJECT;
16         clean(requiredResource);
17     end
18 end
19 else
       /* party reject our counter: finish process.           */
20     clean(requiredResource);
21 end
```

and strategy-proof negotiation [21] can be supported by the Cloud Coordinator if the negotiation Engine is extended with these extra functionalities or if a new Negotiation Engine, supporting such features, replaces the one described in this section.

4.4. Market Engine

Selling price of resources is determined by the *Market Engine* component of the Cloud Coordinator. It is implemented as an abstract class, so different pricing strategies can be easily added. Currently, pricing is static and defined in a configuration file. We also implemented a pricing mechanism similar to the one applied by Amazon EC2, where prices are based on the characteristics of the VM, considering a finite number of available specifications.

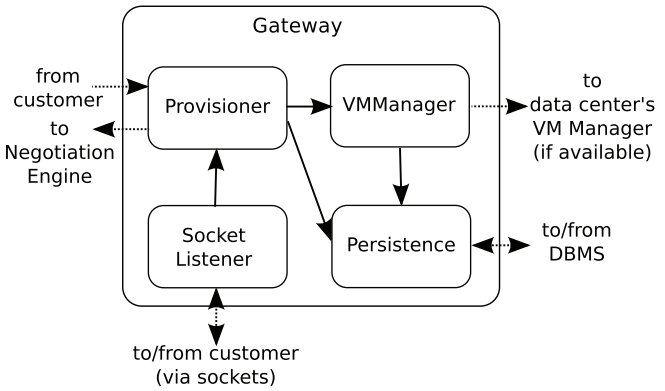


Fig. 5. Gateway organization.

However, the design of the component allows new strategies, such as dynamic pricing based on resources availability, to be added. This is desirable, because more complex policies may contribute for increasing providers' revenue. For example, Toosi et al. [7] showed that pricing policies for federated Clouds based on local resources availability may increase provider's profit and resource utilization. By applying these types of policies in InterCloud, it is expected that providers will have financial motivation to make resources available in the market.

4.5. Gateway

The Gateway is the component that intermediates the access to the local physical infrastructure. If the Cloud Coordinator works as a broker only, e.g., there is no local resources to be managed, the Gateway is still responsible for receiving user requests and triggering requests for resources in the market. Gateway organization and class diagram are presented in Figs. 5 and 6 respectively.

Access to physical resources are abstracted by the *VMManager* component. It exposes an interface related to VM management, including mapping, creation, and destruction of VMs. It also implements the operations that are not offered by the actual VM manager, while it delegates to the latter the operation the latter supports. This enables relevant management services to be developed independently from the capabilities of available virtual machine managers, and connectors translate the requests to the specific manager interface. Current VM managers supported by InterCloud are Eucalyptus [34], OpenNebula [35], Aneka [36], and Amazon EC2.[4]

Besides access to physical resources, Gateway also receives local customer requests and serves them. When a request arrives, the *Provisioner* checks with the VMManager if the request can be served locally. If so, the request is forwarded to the *VMManager* and VMs are allocated accordingly to the request. If not, Provisioner forwards the request to the Negotiation Engine and resources for the request are sought in the InterCloud market, as previously described.

Another component of the Gateway is the *Persistence* module, which is responsible for keeping information about allowed types of virtual machines that can be deployed in the infrastructure, as

[4] In this case, resources are not actually available inside the data center, but they are acquired from Amazon on demand.

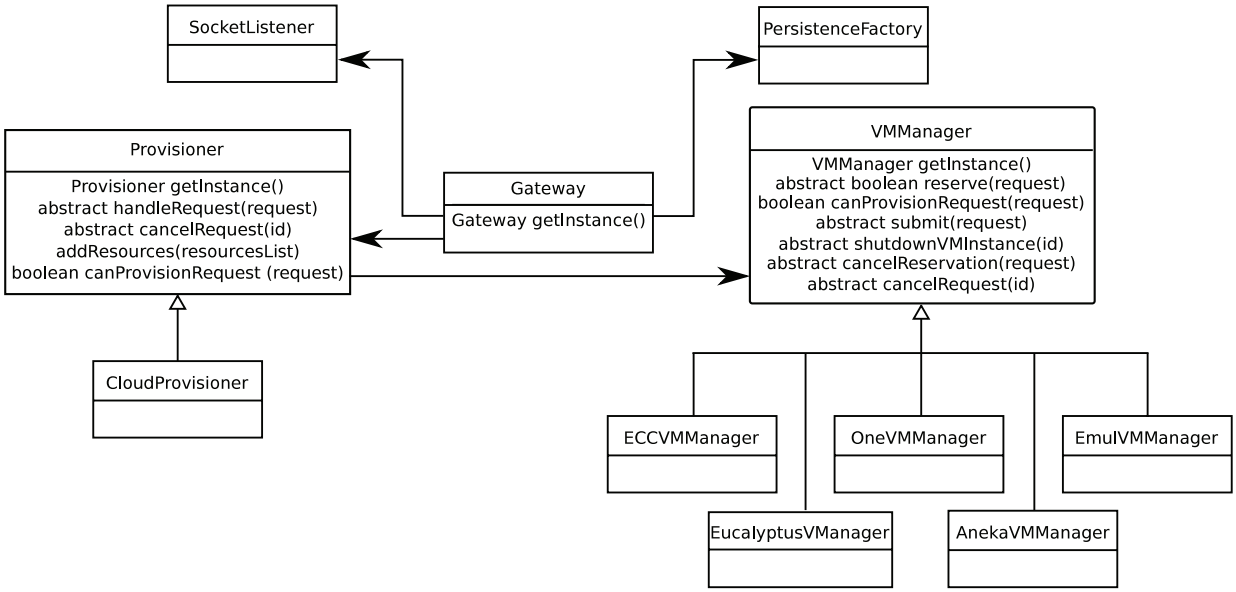


Fig. 6. Gateway class diagram, containing its main supporting classes and their public methods.

well as keeping track of VM images from users. Finally, a *Socket Listener* module allows sending and receiving of customer requests and its response via sockets.

4.6. Resources trading process

In this section, we describe the process for resource trading currently in place in the Cloud Coordinator. Such a process of resource trading involves three main aspects: (i) when process for resource selling starts; (ii) when process for resource buying starts; and (iii) when resources are successfully acquired.

Regarding decision about selling resources to InterCloud members, all the idle resources in the infrastructure are made available to members. Providers update such information in the Cloud Exchange every 30 min, unless availability has not changed in the period.

Process for resource buying starts when the provider receives a request that cannot be served by local resources. The policy is implemented as part of the Negotiation Engine and makes two assumptions: the first one is that customers do not accept a partially served request. It means that the exact number of resources has to be made available to a customer in response to their request, or the request is rejected.

The second assumption is that a request has to be entirely composed of resources from the same data center. This is because IaaS providers typically charge customers by network traffic to and from the data center, but they do not charge for internal traffic. Therefore, customers may assume, when designing and deploying their applications in the infrastructure, that traffic will be local and therefore free of charge and with smaller latency.

Nevertheless, each request for resources for the same elastic application is treated as an independent request, and thus it can eventually be provisioned in a different location from the other resources available for such an application. Thus, for example, consider a request for 10 VMs for an elastic application A that was provisioned by a provider P_1 in its own data center. If later the application has to be scaled up with more 5 VMs, a new request for 5 VMs is sent to P_1, and these 5 VMs may either be created in P_1's data center or have to be created in another InterCloud data center able to support all the five VMs.

When a request that cannot be entirely served in the provider's data center arrives, the allocation policy process takes place. Initially, a request for compatible resources is sent to the Cloud Exchange. Among all the available offers that match the requirement, the chosen one is the cheapest one that meets user VM requirements. However, if the unit cost per hour per VM for using InterCloud machines is bigger than the unit cost paid by the customer, the negotiation process is not held. As already discussed, pricing information given by the Cloud Exchange is used only for creating the availability list and order the preferred resource suppliers, because availability may have changed since its last update. Negotiation runs between the two Cloud Coordinators regardless of the price published by the Cloud Exchange. However, if the price is above the maximum price the buyer would pay, negotiation is canceled and the next provider is queried.

With the above policy, an IaaS provider surrenders its profit from the specific customer request to avoid risking losing its reputation or losing the customer to another provider. This policy can be easily adapted to consider operational costs when deciding whether InterCloud resources will be bought or not. In this case, the maximum amount a provider pays for InterCloud resources are the unit price per resource payed by the customer minus the cost of handling the request for such a resource.

5. Performance evaluation

In this section, we present the evaluation of our Cloud Coordinator prototype. The section starts with the description of the small-scale InterCloud infrastructure where the prototype was deployed and executed. Then three experiments, showing respectively interaction between components, evaluation of mechanisms, and impact on elastic applications, are presented.

5.1. Experimental setup

Fig. 7 depicts the physical infrastructure employed in the experiments. This is composed of one Cloud Exchange and two Cloud Coordinators. The Cloud Coordinators are in a local area network in Melbourne, Australia. One of such Cloud Coordinators (referred to as A in the rest of this section) manages a physical

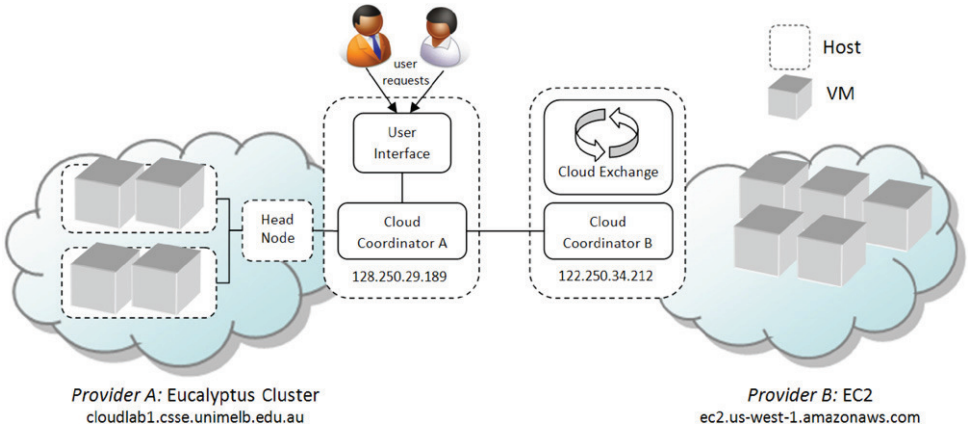


Fig. 7. Scenario used in the experiments.

infrastructure comprised of three Xeon Quad Core 2.00 GHZ processors with 8 GB of RAM and two 160 GB SATA disks (mirrored RAID 1) each. Management of virtual machines on such resources is carried out by Eucalyptus 1.6.2 that runs in one of the servers, while the other two servers are nodes for hosting VMs. The second Cloud Coordinator (referred to as B in the rest of this section) mediates access to Amazon EC2 resources located in USA west coast. Virtual resources in both data centers are offered and charged accordingly to Amazon EC2 pricing policy.

For the sake of simplicity, we allow in these experiments only three types of instances: Small (1 core, 1.7 GB of RAM and 1 EC2 Computing Unit, costing $0.095 per instance per hour), Large (2 cores with 2 EC2 Computing Units each, 7.5 GB of RAM, costing $0.38 per instance per hour), and Extra Large (4 cores with 2 EC2 Computing Units each, 15 GB of RAM, costing $0.76 per instance per hour). In Data Center B we permit only creation of up to 10 instances, regardless of their type. In Data Center A we allow as many instances as supported by the cluster nodes (4 Small or 1 Large on each node). All the requests are generated in Provider A.

Web services (both Cloud Coordinators and Cloud Exchange) are deployed in GlassFish v3 application servers. Both the Cloud Exchange and the Cloud Coordinator B run in an Intel Core2 6600 (Dual Core, 2.4 GHz) with 2 GB of RAM and 70 GB of disk. The Cloud Coordinator A runs in an Intel Core2 Duo E8400 (Dual Core, 3 GHz) with 3 GB of RAM and 140 GB of disk.

5.2. Experiment 1: interaction with InterCloud components

The first experiment aims at demonstrating that our prototype Cloud Coordinator interacts with the other InterCloud components accordingly with the model described in the previous sections. To show that, we show that Cloud Coordinators are able to submit offers and requests to the Cloud Exchange.

When the Cloud Coordinator initializes, it sends to Exchange information about available resources and their characteristics. Therefore, to check if this information was actually published, we send a *getCurrentOffers* request to the Cloud Exchange, via Glassfish's web server management interface. Such a request has the form {vms = 1, memory = 1, power = 1.0, price = $100.0}. With such a request for minimum amounts of each resource and with a high maximum price, all the requests published in the Exchange should be returned. The SOAP request sent to the Cloud Exchange and the response of such a request are shown in Fig. 8.

The Cloud Exchange response, which is presented in Table 3, shows the offers published by each Cloud Coordinator: as expected, both Cloud Coordinators published correct values of availability, characteristics, and prices, as defined in the experiment set up. It shows that Cloud Coordinators are able to correctly advertise their available resources in the Cloud Exchange and provide their endpoint addresses to potential resource buyers.

5.3. Experiment 2: discovery, negotiation, and provisioning

This experiment aims at demonstrating the discovery, negotiation, and resource provisioning mechanisms of InterCloud in action, with providers negotiating for resources to meet internal demand or increase utilization of local resources. This experiment uses the same set up from the previous experiment.

After the deployment of Cloud Coordinators and publication of available resources in the Cloud Exchange (showed in the previous experiment), user requests were submitted to Provider A. The first user request sent to the provider demanded four Small instances. A short time later, a second request, demanding five Small instances, is sent to the same provider. The response users receive from such requests is a list containing addresses of the required VMs.

To demonstrate how the allocation process is realized, Fig. 9(a) contains the screenshot of the management interface of the Eucalyptus cluster. The figure shows that the first request could be served in the provider that received the request. Therefore, by inspecting the Eucalyptus cluster management interface we can see that four machines configured as Amazon's Small instances were allocated from the cluster. The second request, however, could not be completely served in the local cluster and thus resources for such request were acquired from other CloudCoordinators. Maximum cost to be payed by such extra resources was the same as the price paid by local resources, which was $0.095.

Because, according to the Cloud Exchange response, there was a provider able to serve such a demand (Provider B), negotiation started between the two providers. Because no request was submitted to Provider B, its full capacity was still available to InterCloud members. Moreover, price payed by customers on both providers are the same, so negotiation was successful. As a result, five Small instances were created in Amazon EC2 in response to the request of Provider A. Such instances can be seen in Fig. 9(b).

Table 4 shows relevant operations executed during the experiment and time taken for each of them to complete. This information was obtained from execution logs generated by each Cloud Coordinator. It shows that the Cloud Coordinator initialization takes in average 6.8 s. This time is necessary to start the web services engine and initialize the Gateway. Gateway

a

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:getCurrentOffers xmlns:ns2="http://market.intercloud.cloudbus.org/">
            <arg0>1</arg0>
            <arg1>1</arg1>
            <arg2>1</arg2>
            <arg3>1.0</arg3>
            <arg4>100.0</arg4>
        </ns2:getCurrentOffers>
    </S:Body>
</S:Envelope>
```

b

SOAP Response

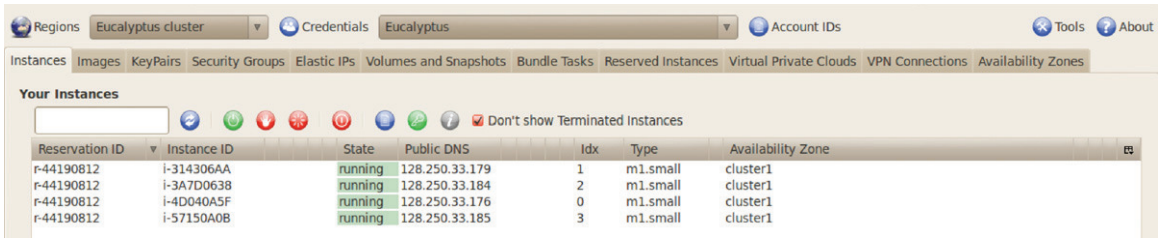
```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:getCurrentOffersResponse xmlns:ns2="http://market.intercloud.cloudbus.org/">
            <return>http://128.250.34.212:8080/InterCloudWS/CloudCoordinatorService#1740#1#10#1.0#0.095</return>
            <return>http://128.250.29.189:8080/InterCloudWS/CloudCoordinatorService#1740#1#8#1.0#0.095</return>
            <return>http://128.250.34.212:8080/InterCloudWS/CloudCoordinatorService#7680#2#10#2.0#0.38</return>
            <return>http://128.250.29.189:8080/InterCloudWS/CloudCoordinatorService#7680#2#2#2.0#0.38</return>
            <return>http://128.250.34.212:8080/InterCloudWS/CloudCoordinatorService#15360#4#10#2.0#0.76</return>
        </ns2:getCurrentOffersResponse>
    </S:Body>
</S:Envelope>
```

Fig. 8. SOAP (a) request and (b) response to a *getCurrentOffers* request to the Cloud Exchange, after deployment of both Cloud Coordinators and before execution of requests from users. They were obtained from a screenshot of the administration console of Glassfish.

Table 3
Response from the Cloud Exchange about available resources. Endpoint addresses presented in the table have been shortened in comparison to the actual return from the Cloud Exchange (shown in Fig. 8). Power is given in EC2 Compute Units.

Endpoint	Memory (MB)	VMs	Cores	Power	Price (U$)
http://128.250.34.212:8080...	1 740	1	10	1.0	0.095
http://128.250.29.189:8080...	1 740	1	8	1.0	0.095
http://128.250.34.212:8080...	7 680	2	10	2.0	0.38
http://128.250.29.189:8080...	7 680	2	2	2.0	0.38
http://128.250.34.212:8080...	15 360	4	10	2.0	0.76

a



b

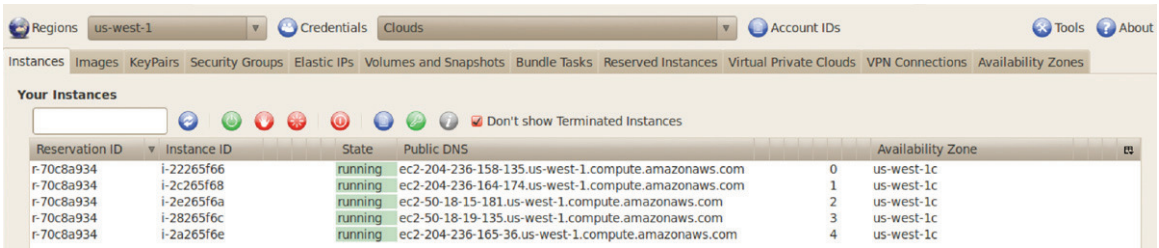


Fig. 9. Management output of (a) Eucalyptus cluster and (b) Amazon EC2, showing the allocation of resources during the experiments.

Table 4
Performance of relevant operations executed during experiments.

Operation	Time (ms)
Initialization of Cloud Coordinators (average)	6 803
Deployment of VMs in the Eucalyptus cluster	4 999
Time between reception of first user request and return of response	8 850
Deployment of VMs in Amazon EC2	16 695
Time between reception of second user request and return of response	16 816

initialization, on the other hand, requires initialization of the Derby DB, VM Manager, and other components.

Deployment time is computed from the moment the deployment process of VMs starts until the addresses of the required VMs are available to customers. It does not mean that VMs are ready to be used after this time, because VMs can still be initializing. Nevertheless, as their addresses are already known by this time, they are returned to customers and the request is considered as served. Deployment time in Amazon was bigger because it required communication with the Amazon data center located in North America. This also makes the total response time of the second request, which was served by Amazon resources, longer than the response time of the first request, which was served by local resources.

These results show that the discovery, negotiation, and provisioning processes proposed in the InterCloud mode, and implemented in our prototype, are feasible and allows IaaS providers to serve more requests than their local infrastructure alone allows.

5.4. Experiment 3: impact of InterCloud on elastic applications

This experiment aims at evaluating the impact of InterCloud on elastic applications. The elastic application used in the experiment is a Bag-of-Tasks (BoT) version of the EMO (Evolutionary Multi-objective Optimizer) multi-objective evolutionary optimizer based on genetic algorithms [36]. The optimization problem solved in this experiments was DTLZ6 [37]. The corresponding BoT job consists of 60 tasks, where each tasks resolves the problem using 400 generations and 150 individuals.

To allow EMO to run on elastic infrastructures, Aneka [38] has been used as the client of a Cloud provider. Aneka provisions resources to the application by acquiring resources from the InterCloud gateway it is connected to. Notice that dynamic provisioning capabilities of Aneka could be used to allow acquisition of resources of multiple sites; however, such an approach would require previous configuration of Aneka to support and locate such providers. By accessing multiple providers via InterCloud, Aneka is able to deliver resources to application without (i) knowledge of specific providers location, (ii) implementing APIs to access such providers; and (iii) undertaking negotiation for resources. Therefore, the whole process for provisioning is simplified in the application side: instead of adding support for each provider in InterCloud, Aneka just communicates with a local InterCloud gateway to request resources.

Integration between Aneka and InterCloud was achieved by configuring a *InterCloud Resource Pool* in Aneka. Such a resource pool transforms one Aneka's request for n resources in n requests for one resource that are sent to InterCloud. Such request transformation is possible because the test application is BoT, thus it is not necessary that all the tasks run in the same provider. If a single request for n resources were sent to InterCloud, only a single provider able to deliver the n VMs would be used.

The scenario on which the experiment run is similar to the scenario presented in Fig. 7, except that the Cloud Coordinator A has been replaced by a coordinator that mediates access to Amazon EC2 resources located on USA east coast. Likewise the restriction imposed to the Cloud Coordinator B in US west coast, the new Cloud Coordinator A can allocate a maximum of 10 VMs. Virtual machines run Ubuntu 10.10 and contain Mono version 2, which is required by Aneka worker nodes that execute the application. Aneka master runs in a machine with Windows operating systems and its requests for resources are forwarded to the Cloud Coordinator A.

To explore application elasticity, different execution time targets were set in Aneka for the application. These targets are used as guidelines for Aneka's scheduler and resource provisioner to calculate the required number of VMs. This calculation ignores time for VM creation and data transferring, and thus the actual execution time tends to be higher than the target if external resources (resources that are not directly managed by Aneka) are used. Considering that in our experiment there is no internal resources to run the application, a different number of machines are requested by Aneka to the Cloud Coordinator on each execution of the application in order to meet the target.

Following the policies described in previous sections, resources are preferentially supplied by the provider receiving the request, and when local resources are not enough, resources from Cloud Coordinator B are requested via Cloud Exchange. For the cases where InterCloud resources were required, we give also the minimum execution time that can be achieved without these extra resources, which is the execution time obtained when all the resources from Provider A were used. This allowed us to evaluate the impact of InterCloud in the application execution time. Table 5 presents the results of application execution considering different application targets.

Besides the speed up in the application execution time and the capacity of meeting shorter deadlines with the use of InterCloud resources, the experiment also shows the qualitative benefit of allowing the application to seamlessly access resources from different providers, because from the point of view of the application all the resources were supplied by the same provider. Therefore, the burden of discovering providers able to serve the request and negotiating for such resources are removed from the application level, allowing user-level software to focus on the user application.

6. Conclusions and future directions

The goal of InterCloud is enabling seamless meet of elastic applications' SLAs by scaling them across various data centers, improving applications' performance, reliability, and scalability.

In this paper, we presented the architecture, design, and evaluation of the Cloud Coordinator element from the InterCloud architecture. The Cloud Coordinator represents data centers and brokers in the InterCloud marketplace, and it is responsible for publishing offers and requests for resources, discovering potential providers of resources, and negotiating resources when it is necessary. In its backend, it interacts with the virtual machine manager from the local infrastructure, so it can be advised about the need for extra resources and supply them to the provider. A prototype was developed and deployed in a small-scale InterCloud scenario. Evaluation of the prototype showed

Table 5
Results of the execution of the test elastic application in InterCloud. The "Target" column, an application parameter, describes the target runtime of the application excluding delays caused by VM initialization and network transfers. This value is supplied by the Aneka user together with an estimation of application runtime. Tests without InterCloud consider utilization of resources available in Provider A only.

Target (min)	Utilized VMs		Execution time	
	Provider A	Provider B	Without InterCloud	With InterCloud
45	7	0	50 min 15 s	
35	9	0	43 min 32 s	
25	10	1	36 min 10 s	34 min 10 s
15	10	7		24 min 03 s

the effectiveness of the proposed approach for the envisioned scenario.

As part of future work, we plan to develop policies for decision on buying and selling resources, as well as policies for dynamic pricing for resources. Moreover, aspects such as advanced reservation and co-allocation will be considered. We will also work into prediction of demand, so that resources can be acquired short before they are necessary, in order to improve application performance by hiding overhead of virtual machine creation and initialization.

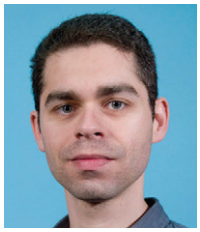
Acknowledgments

Authors wish to thank Dileban Karunamoorthy and Suraj Pandey for their support in the experiments and Sivaram Yoganathan for his insightful comments and proofreading of the paper.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.
[2] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 cloud computing services for scientific computing, in: Proceedings of the 1st International Conference on Cloud Computing, CloudComp'09, Springer, Munich, Germany, 2009, pp. 115–131.
[3] J. Dejun, G. Pierre, C.-H. Chi, Resource provisioning of web applications in heterogeneous clouds, in: Proceedings of the 2nd USENIX Conference on Web Application Development, WebApps'11, USENIX, Portland, USA, 2011.
[4] R. Buyya, R. Ranjan, R.N. Calheiros, InterCloud: utility-oriented federation of cloud computing environments for scaling of application services, in: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP'10, Springer, Busan, South Korea, 2010, pp. 13–31.
[5] S.K. Garg, C. Vecchiola, R. Buyya, Mandi: a market exchange for trading utility and cloud computing services, The Journal of Supercomputing (2011) http://dx.doi.org/10.1007/s11227-011-0568-6.
[6] S. Venugopal, X. Chu, R. Buyya, A negotiation mechanism for advance resource reservations using the alternate offers protocol, in: Proceedings of the 16th International Workshop on Quality of Service, IWQoS'08, IEEE Computer Society, Enschede, Netherlands, 2008, pp. 40–49.
[7] A.N. Toosi, R.N. Calheiros, R.K. Thulasiram, R. Buyya, Resource provisioning policies to increase IaaS provider's profit in a federated cloud environment, in: Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications, HPCC'11, IEEE Computer Society, Banff, Canada, 2011.
[8] K.M. Sim, Agent-based cloud commerce, in: Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, IEEM, IEEE, Hong Kong, 2009, pp. 717–721.
[9] I. Foster, C. Kesselman (Eds.), The Grid 2: Blueprint for a New Computing Infrastructure, second ed., Morgan Kaufmann, 2003.
[10] A. di Costanzo, M.D. de Assunção, R. Buyya, Harnessing cloud technologies for a virtualized distributed computing infrastructure, IEEE Internet Computing 13 (5) (2009) 24–33.
[11] B. Rochwerger, et al., Reservoir—when one cloud is not enough, Computer 44 (3) (2011) 44–51.
[12] L. Rodero-Merino, L.M. Vaquero, V. Gil, F. Galán, J. Fontán, R.S. Montero, I.M. Llorente, From infrastructure delivery to service management in clouds, Future Generation Computer Systems 26 (8) (2010) 1226–1240.
[13] A.I. Avetisyan, et al., Open Cirrus: a global cloud computing testbed, Computer 43 (4) (2010) 35–43.
[14] K. Keahey, M. Tsugawa, A. Matsunaga, J.A.B. Fortes, Sky computing, IEEE Internet Computing 13 (5) (2009) 43–51.
[15] A.J. Ferrer, et al., Optimis: a holistic approach to cloud service provisioning, Future Generation Computer Systems 28 (1) (2012) 66–77.
[16] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, M. Morrow, Blueprint for the InterCloud—protocols and formats for cloud computing interoperability, in: Proceedings of the 4th International Conference on Internet and Web Applications and Services, ICIW'09, IARIA, Venice, Italy, 2009, pp. 328–336.
[17] A. Celesti, F. Tusa, M. Villari, A. Puliafito, How to enhance cloud architectures to enable cross-federation, in: Proceedings of the 3rd International Conference on Cloud Computing, CLOUD'10, IEEE Computer Society, Miami, USA, 2010, pp. 337–345.
[18] J. Altmann, C. Courcoubetis, M. Risch, A marketplace and its market mechanism for trading commoditized computing resources, Annals of Telecommunications 65 (11) (2010) 653–667.
[19] M. Macías, O. Rana, G. Smith, J. Guitart, J. Torres, Maximizing revenue in grid markets using an economically enhanced resource manager, Concurrency and Computation: Practice and Experience 22 (14) (2010) 1990–2011.
[20] B. Song, M.M. Hassan, E.-N. Huh, A novel cloud market infrastructure for trading service, in: Proceedings of the 9th International Conference on ComputationalScience and its Applications, ICCSA'09, IEEE Computer Society, Suwon, South Korea, 2009, pp. 44–50.
[21] M. Mihailescu, Y.M. Teo, Strategy-proof dynamic resource pricing of multiple resource types on federated clouds, in: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP'10, Springer, Busan, South Korea, 2010, pp. 337–350.
[22] E.R. Gomes, Q.B. Vo, R. Kowalczyk, Pure exchange markets for resource sharing in federated clouds, Concurrency and Computation: Practice and Experience (2010) http://dx.doi.org/10.1002/cpe.1659.
[23] K. Vanmechelen, W. Depoorter, J. Broeckhove, Combining futures and spot markets: a hybrid market approach to economic grid resource management, Journal of Grid Computing 9 (1) (2011) 81–94.
[24] H. Kim, Y. el Khamra, S. Jha, M. Parashar, Exploring application and infrastructure adaptation on hybrid grid-cloudinfrastructure, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10, ACM, Chicago, USA, 2010, pp. 402–412.
[25] S. Ostermann, R. Prodan, T. Fahringer, Extending grids with cloud resource management for scientific computing, in: Proceedings of the 10th IEEE/ACM International Conference on Grid Computing, GRID'09, IEEE Computer Society, Banff, Canada, 2009, pp. 42–49.
[26] M.D. de Assunção, A. di Costanzo, R. Buyya, Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, in: Proceedings of the 18th International Symposium on High Performance Distributed Computing, HPDC'09, ACM, Munich, Germany, 2009, pp. 141–150.
[27] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, Future Generation Computer Systems 28 (2) (2012) 358–367.
[28] R.V. den Bossche, K. Vanmechelen, J. Broeckhove, Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads, in: Proceedings of the 3rd International Conference on Cloud Computing, CLOUD'10, IEEE Computer Society, Miami, USA, 2010, pp. 228–235.
[29] J.O. Fitó, Í. Goiri, J. Guitart, SLA-driven elastic cloud hosting provider, in: Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP'10, IEEE Computer Society, Pisa, Italy, 2010, pp. 111–118.
[30] Y.C. Lee, C. Wang, A.Y. Zomaya, B.B. Zhou, Profit-driven service request scheduling in clouds, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid'10, IEEE Computer Society, Melbourne, Australia, 2010, pp. 15–24.
[31] Í. Goiri, J. Guitart, J. Torres, Characterizing cloud federation for enhancing providers' profit, in: Proceedings of the 3rd International Conference on Cloud Computing, CLOUD'10, IEEE Computer Society, Miami, USA, 2010, pp. 123–130.
[32] J. Varia, Best practices in architecting cloud applications in the AWS cloud, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press, New Jersey, USA, 2011, pp. 459–490. (Chapter 18).
[33] DMTF, Open virtualization format specification version 1.1.0, DMTF Standard, 2010. URL: http://www.dmtf.org/standards/ovf.

[34] D. Nurmi, et al., The Eucalyptus open-source cloud computing system, in: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid'09, IEEE Computer Society, Shanghai, China, 2009, pp. 124–131.
[35] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing 13 (5) (2009) 14–22.
[36] C. Vecchiola, X. Chu, R. Buyya, Aneka: a software platform for.NET-based cloud computing, in: W. Gentzsch, L. Grandinetti, G. Joubert (Eds.), High Performance and Large Scale Scientific Computing, IOS Press, Amsterdam, Netherlands, 2009.
[37] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable test problems for evolutionary multiobjective optimization, in: A. Abraham, L. Jain, R. Goldberg (Eds.), Evolutionary Multiobjective Optimization: Theoretical Advances and Applications, Springer, London, UK, 2005.
[38] C. Vecchiola, M. Kirley, R. Buyya, Multi-objective problem solving with offspring on enterprise clouds, in: Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region, HPCAsia'09, NCHC, Kaohsiung, Taiwan, 2009, pp. 132–139.



Rodrigo N. Calheiros is a Research Fellow in the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Dept. of Computing and Information Systems, University of Melbourne, Australia. He completed his Ph.D. degree in Computer Science in 2010 at PUCRS, Brazil, and his M.Sc. degree in 2006 at the same University. His research interests include Cloud Computing and simulation and emulation of distributed systems, with emphasis in Grids and Clouds.

Adel Nadjaran Toosi is a Ph.D. student at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia. He received his B.Sc. degree in 2003 and his M.Sc. degree in 2006 both in Computer Software Engineering from the Ferdowsi University of Mashhad, Iran. His research interests include Distributed System, Cloud Computing, Cloud Federation and InterCloud. Currently, he is working on economic aspects of the InterCloud project, a framework for federated Cloud Computing.

Christian Vecchiola is a Research Fellow at the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Department of Computing and Information Systems, at The University of Melbourne, Australia. His primary research interests include Grid/Cloud Computing, Distributed Evolutionary Computation, and Software Engineering. Since he joined the CLOUDS Lab he focused his research activities and development efforts on two major topics: middleware support for Cloud/Grid Computing and distributed support for evolutionary algorithms.

Christian completed his Ph.D. in 2007 at the University of Genova, Italy with a thesis on providing support for evolvable Software Systems by using Agent Oriented Software Engineering. During the Ph.D. he worked under the supervision of Prof. Antonio Boccalatte in the Department of Communication Computer and System Sciences and has have been actively involved in the design and the development of the AgentService that is a software framework for developing distributed systems based on Agent Technology. Dr. Vecchiola also investigated the advantages of providing support for agent based development at a programming language level by extending the object oriented language with abstractions for representing the key elements of the agent computing model.

Rajkumar Buyya is a Professor of Computer Science and Software Engineering and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored 350 publications and four text books. He also edited several books including "Cloud Computing: Principles and Paradigms" recently published by Wiley Press, USA. He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 53, g-index = 114, 15 000 + citations).

Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. These contributions and international research leadership of Dr. Buyya are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society, USA. Manjrasofts Aneka Cloud technology developed under his leadership has received "2010 Asia Pacific Frost & Sullivan New Product Innovation Award".